Evaluation of the DSN Software Methodology

A. Irvine
TDA Engineering Office

M. McKenzie
TDA Planning Office

This article describes the effects of the DSN software methodology, as implemented under the DSN Programming System, on the DSN Mark III Data Subsystems Implementation Project (MDS). The software methodology is found to provide a markedly increased visibility to management, and to produce software of greater reliability at a small decrease in implementation cost. It is also projected that additional savings will result during the maintenance phase. Documentation support is identified as an area that is receiving further attention.

I. Introduction

This article describes the quantitative and qualitative effects of the software methodology (Ref. 1) as implemented under the DSN Programming System, on the DSN Mark III Data Subsystems Implementation Project (MDS). The period in the life cycle of the MDS that was studied extended from September 1974 through November 1976. During this period, approximately 100,000 lines of application code were delivered as well as diagnostic and system software.

The scope of this article covers the implementation of the approximately 100,000 lines of application code that can be directly allocated to the following subsystems:

DCD	Command
DTM	Telemetry
DTK	Tracking
DMC	Monitor
DTT	Test and Training
CMF	Communications Monitor and Formatter
DST	Host

The subsystems are broken down by size in Table 1.

The effects of the methodology on a single subsystem were studied in detail and the results, with corroboration from a second subsystem, were extrapolated for the entire effort. In addition to this quantitative evaluation of the MDS implementation, a qualitative evaluation by cognizant software engineers is included at the end of the report.

II. Software Methodology

At the start of the MDS Project most of the Standard Practices governing software implementation had been formulated but were as yet unpublished. However, the concepts governing the methodology were well known and interim documents were published to guide software implementation. The principal concepts used in the MDS software effort and implied by the term "methodology" in this report are:

(1) Structured Programming.

(2) Top-down Development.

(3) Visibility Reporting.

(4) Management Reviews.

(5) Documentation.

III. Impact of Methodology on the Command Subsystem

The Deep Space Station Command Subsystem (DCD) was cnosen as a representative system for the MDS for the following reasons:

(1) The DCD software was of the "average" size of the MDS subsystem software.

(2) The DCD was the best documented subsystem in terms of statistics gathered during the project lifetime under study.

(3) The DCD development team most closely adhered to the methodology.

Several important milestones should be noted that occurred during the Command subsystem implementation and represented application of the methodology. These milestones were:

The Software Requirements Document (SRD) published for DCD	3/3/75
The Software Definition Document (SDD) published for DCD	9/19/78
DCD High-Level Design Review	11/13/75
MDS System Level Review	3/13/76
Work Breakdown Structure (WBS) Methodology applied	4/1/76
Monthly Management Reporting started	5/1/76
Weekly Reports	9/8/76

The impact of these milestones will be discussed.

A. Visibility

Application of the methodology resulted in a series of milestones that were designed to increase management visibility. Visibility was indeed increased; an example is found in the management of the subsystem delivery date. The SRD reported an estimated delivery date of 1 August 1976. By the time the SDD was published, six months later, the estimated delivery date had slipped one month. Six months later, at the MDS System Level Review, a further slip of two months for the estimated delivery date had occurred. The successive slips of estimated delivery date can be projected (see Fig. 1). This projection is nonlinear and, as the desired delivery date is approached, the number of slips tends to increase. This slip in estimated delivery date was typical for all subsystems, but is illustrated only for the DCD.

It was possible because of the visibility to determine seven months in advance of the need date that the actual delivery date of the software was slipping. Increased visibility plus the application of the WBS methodology allowed the splitting of the effort into two phases. The software necessary to meet first priority requirements was identified and that portion of the software was delivered as a first phase, thereby pulling the delivery date back to the need date. Without corrective action, delivery would have taken place four to five months late.

B. Anomalies

Anomalies were not originally reported for all subsystems and original published anomaly reports do not reflect the DTT or DST subsystems. For the MDS, approximately 200 anomalies were reported at verification time and during acceptance testing at CTA 21. After acceptance testing, approximately 90 more were uncovered.

For the DCD, 77 anomalies were reported during verification testing and acceptance testing, and 43 after acceptance testing. While the anomaly curve for the DCD presents the typical curve exhibited for the MDS (Figs. 2 and 3), it can easily be determined that the rate is approximately twice that for the project as a whole.

Possible causes for this phenomenon could be:

(1) The software was not ready for delivery.

(2) Reporting of preacceptance test anomalies was more comprehensive than for other subsystems.

(3) The software was more difficult.

(4) The software was considered more crucial to flight operations and therefore more rigorously tested.

(5) The methodology does not affect the number of anomalies.

Close investigation shows that reasons (2) and (4) are probably the true reasons for this phenomenon. It has already been mentioned that the DCD was the best documented subsystem and team members adhered most closely to the methodology, and therefore were more meticulous about reporting errors during the verification phase. Extensive performance test programs were written that exercised the software extensively.

Even at the rate of 4.8 errors per 1000 lines of code prior to delivery, the DCD software compares favorably to the industry average of 18 per 1000 lines of code (see Refs. 2, 3, 4).

C. Implementation Cost

One measure of the effectiveness of a programming methodology is the productivity rate in delivered lines of source code per man-month (LOC/mm). An industrial average productivity rate is available for projects produced without the elements of the software methodology. For a comparison, it is desirable to determine the corresponding productivity rate for the Command Subsystem.

It is felt that for a fair comparison, only equivalent activities should be included in the productivity measure. Thus, included in the Command manpower measure is the direct effort associated with technical manpower, derived not from the System for Resource Management (SRM), but as a direct translation of time expended by personnel in the various tasks associated with software implementation such as designing, coding, testing, and documentation. The time spent by engineering personnel on documentation is included but not the costs of support people, reproduction, and other costs involved in the production of the documents themselves. Time expended was derived primarily from the DCD Operational Software Development Schedule and Manpower Allocation Chart.

Figure 4 represents the manpower loading of the DCD during the time span under discussion. Studies of available data from industry (Refs. 5 and 6) indicate that an average of 3.2 persons per month (200 LOC/mm) should have been utilized to produce the necessary code. Instead only an average of 2.7 (237 LOC/mm) persons were employed. Over the 25-month period, a cost savings of 1 person was achieved, or approximately $50,000 for one subsystem.

A second subsystem was studied to corroborate the productivity gains found in the Command Subsystem. The Communications Monitor and Formatter (CMF) was chosen and the CMF man-power loading is shown in Fig. 5. Again, based on an industry productivity average of 200 lines of code per month (Refs. 5 and 6), the savings were 17.3 man months.

The increase in productivity can be considered significant for both subsystems because it exceeds one standard deviation from the mean. The result reinforces the belief that small but significant direct implementation cost savings were obtained with the use of the software methodology.

IV. Software Development Period

The development period of 25 months used to develop cost estimates is an extremely conservative one consisting of starting where the first known manpower could be attributed to the project. There are many valid and cogent reasons why a later date may be taken. The implementation schedule (Fig. 6) published at the High Level Design Review shows a start date for software implementation at approximately January 1975. If this date is taken as the start of the development period then the following productivity figures can be derived:

DCD	256 LOC/month
CMF	274 LOC/month

The savings would then be:

DCD	17.4 man/months
CMF	23.3 man/months

or approximately $167,000 for just the two subsystems.

V. Documentation

As stated previously, the productivity measure includes engineering time spent on design, code, testing, and documentation, but does not include special documentation support required by the software methodology. For the MDS, documentation support was provided by the Software Production Management and Control (SPMC) center, and is discussed below.

Despite the methodology requirements to produce documentation before or at least concurrent with code, the delivery of formal, validated documentation has consistently lagged behind transfer of the code. This lag has, for many subsystems, been as long as a year and is partially due to the volume of documentation produced.

Walston and Felix of IBM (Ref. 7) predict on the basis of 46 software projects of varied purposes and methodologies, that about 800 pages of documentation is produced for a subsystem the size of Command. The values for one standard deviation are 400 pages and 1250 pages of documentation. The Command Subsystem produced at least 1600 pages (Table 2). This value is higher than the average found by IBM, and indicates that the methodology results in a higher volume of documentation.

Extrapolation of information supplied by DSN Data Systems indicates that documentation support[1] for the DCD cost $69,000. This is slightly more than the productivity gains shown in Section III, and does not include costs incurred after the time-span of this study. Although the documentation cost is negating productivity gains in the implementation phase, these costs should be offset in the maintenance phase. In the absence of hard data, the cost and volume of documentation required by the methodology is being analyzed.

VI. Maintenance Phase

Unfortunately, the maintenance and operations phase of the MDS system has just begun, so there is no quantitative cost data. Nevertheless, two pieces of information indicate that maintenance of the software will be less costly per LOC than previous projects. First, though given extensive testing, the number of anomalies was low. Secondly, another project produced concurrently with the software methodology showed increased software maintainability. There is no reason to expect a decrease in maintainability, and every reason to expect an increase with the use of the software methodology. Thus, it is predicted that there will be additional savings during the maintenance phase of the MDS.

VII. Conclusions from Quantitative Study

The major quantitatively measurable benefits derived from the methodology discussed in this report appear to be threefold:

(1) The ability to deliver software on time by providing greater visibility and allowing management to intervene before projected slips actually impact delivery schedule.

(2) A lower anomaly rate than the reported industry average, which indicates better software and predicts decreased maintenance costs.

(3) Small but significant cost savings in manpower resulting in lower implementation costs. These savings will increase as the documentation problem is addressed.

[1]Documentation support, as used here, is defined as all the support functions performed by the SPMC for the Command Subsystem. Thus, included in the $69,000 figure are the following: documentation production, editing, and reproduction costs; computer operator costs for disk generation, job runs, source code updating, disc preparation, operating system copies, and module assembly; and code support costs such as the maintenance of the code library.

VIII. Qualitative Evaluation—Benefits

In addition to the quantitative evaluation of the software methodology, a qualitative study was performed using interviews and questionnaires of cognizant software individuals. Qualitative, rather than quantitative, evaluations were sought for several reasons. Record-keeping is a part of the methodology, but quantitative data are often unavailable for older projects. It is also difficult to obtain numeric information from people recalling old projects; often those who directed old projects are no longer at JPL, and data from memory may not be sufficiently accurate for a valid comparison. Thus, the best information to be gained from these individuals is qualitative.

A specific questionnaire and open discussion format were used in the interview process. The programmers and managers were also asked to relate their experiences on projects conducted prior to addition of the software methodology, and specify areas in which the methodology would have been beneficial. The results of the interviews show a number of common impressions and experiences, which are discussed below.

The primary problem observed in older software projects was a failure to transfer to operations on schedule. Often accompanying this overrun in schedule was an overrun in budget. With these two conditions, and pressure to transfer as soon as possible, documentation suffered and maintenance was difficult once transfer was achieved. The people interviewed noted a number of causes that produced this condition.

The first was the general difficulty in accurately predicting task complexity, required resources, and the final amount of code. Cost and schedule overruns occurred and some incompatibilities with hardware resulted. This problem not only affected past DSN projects, but still affects the software industry as a whole. Nevertheless, it is felt that the methodology's requirements for modularization and Work Breakdown Structure, with a strong emphasis on design, strongly support better schedule, cost, and resource planning.

A second cause of overruns noted by the software personnel was frequent changes in requirements. This instability disrupted schedules and necessitated numerous releases and transfer liens. The software methodology addresses this problem by formalizing the requirements process and enforcing approval of requirement changes.

Occasionally noted was the condition in which the project was due for transfer and management then learned it was far from complete. Management had not adequately monitored the progress of the project. Again, a benefit of the methodology is that it includes the Work Breakdown Structure, design reviews, and formal milestones, which all address the monitoring of project progress.

The software projects discussed in the interviews all had late and/or insufficient documentation, and major transfer liens were frequent against these documents. Numerous problems resulted: the project was delayed, new personnel found it difficult to work on the software, or testing and debugging required excessive resources. It is felt that the concurrent documentation and design review requirements of the methodology certainly ease these problems. Nevertheless, some documentation is still late. This is partly attributed to flowcharts that sometimes require as long as a month for machine production. By that time, they must often be resubmitted to include changes that occurred during that month. A faster method of documentation production was strongly suggested by the software personnel.

The final cause of project overruns brought out in the interviews was excessive errors in the software. Either major pretransfer testing was required, or numerous bugs were found after transfer. Neither case is desirable. The personnel interviewed felt the modularization, emphasis on design, and formal testing incorporated in the methodology result in less testing, fewer anomolies, and easier debugging.

In summary, projects in the past have often encountered difficulties with schedules and resources, specification changes, management visibility, documentation, and testing. The DSN Software Methodology provides a good structure with which to control these problems.

IX. Qualitative Evaluation— Suggested Improvements

In addition to benefits resulting from the software methodology, the software people suggested the following methodology improvements that would produce even more cost-effective software:

(1) The methodology requirements are occasionally not applicable to a specific project, and obtaining a waiver requires time. Perhaps, with a growing experience in using the methodology, a list of requirements by project size, cost, and type could be developed.

(2) The projects still find themselves caught between documentation requirements and the scheduled transfer date. Part of the problem is felt to lie in the slow turnaround time for finished documentation. Also, the requirement of low-level flowcharting results in a great deal of production and revision time. It is suggested that either (a) only higher-level flowcharting be required, (b) a program design language tool be used in place of flowcharts, or (c) a quick method of flowchart production be found. In all, the software personnel feel that flowcharting currently requires that the programmer cater to the machine, rather than the reverse.

(3) The inclusion of programmers, operators, Cognizant Development Engineers (CDEs), and customers in the requirements and broad design process should be encouraged even more than is currently done.

(4) Too much emphasis is given to the quantity of response to the methodology rather than the quality of the response. An effort should be made to determine levels of quality based on maintainability and error rate after transfer.

(5) The methodology can help standardize the development of software and control the flow of projects, but it cannot force a bad manager or programmer into good performance. A key also lies in the people who perform the job.

References

1. *Standard Practices for the Implementation of Computer Software,* edited by A. P. Irvine, JPL Publication 78-53. Jet Propulsion Laboratory, Pasadena, Calif. September 1, 1978.
2. Michael E. Fagen, *Design and Code Inspections and Process Control in the Development of Programs,* IBM Technical Report TR 21.572, Dec. 17, 1974.
3. J. C. Kickson, et al., "Quantitative Analysis of Software Reliability," *Proc. 1972 IEEE Annual Reliability and Maintainability Symposium,* N.Y., January 1972, pp. 148-157.
4. M. L. Shooman, "Probabilistic Models for Software Reliability Prediction," *Proc. 1972 International Symposium on Fault-Tolerance Computing,* Newton, Mass., June 1972, pp. 211-215.
5. Steve Caine, of Caine, Farber, and Gordon, Inc. Private Communication, Oct. 1976.
6. Stephen R. McCammon, "Applied Software Engineering: A Real-Time Simulator Case History," *IEEE Transactions on Software Engineering,* Vol. SE-1, No. 4, Dec. 1975, p. 383.
7. C. E. Walston and C. P. Felix, "A Method of Programming Measurement and Estimation," *IBM System Journal,* No. 1, 1977, p. 62.

Table 1. Phase I delivered source lines of code

Subsystem		Lines of code
DCD	Command	16,000
DTM	Telemetry	30,100
DTK	Tracking	12,000
DMC	Monitor	16,000
DTT	Test and Training	2,100
CMF	Communications Monitor and Formatter	17,700
DST	Host	5,000
Total lines of code[a]		98,900

[a]The Standard Operating System is not included in the total lines of code. Size of the Standard Operating System is 16,500 lines of code.

Table 2. Amount of documentation for the Command Subsystem

Document	Number of pages
Project Notebook	154
Software Requirements Document	43
Software Definition Document	123
Software Specifications Document[a]	780
Software Operations Manual[a]	140
Software Test and Transfer Document[a]	250
Anomaly Reports	127
Total Documentation	1617 Pages

[a]Amount of documentation was estimated because there was no formal, published document during the time-span under study.

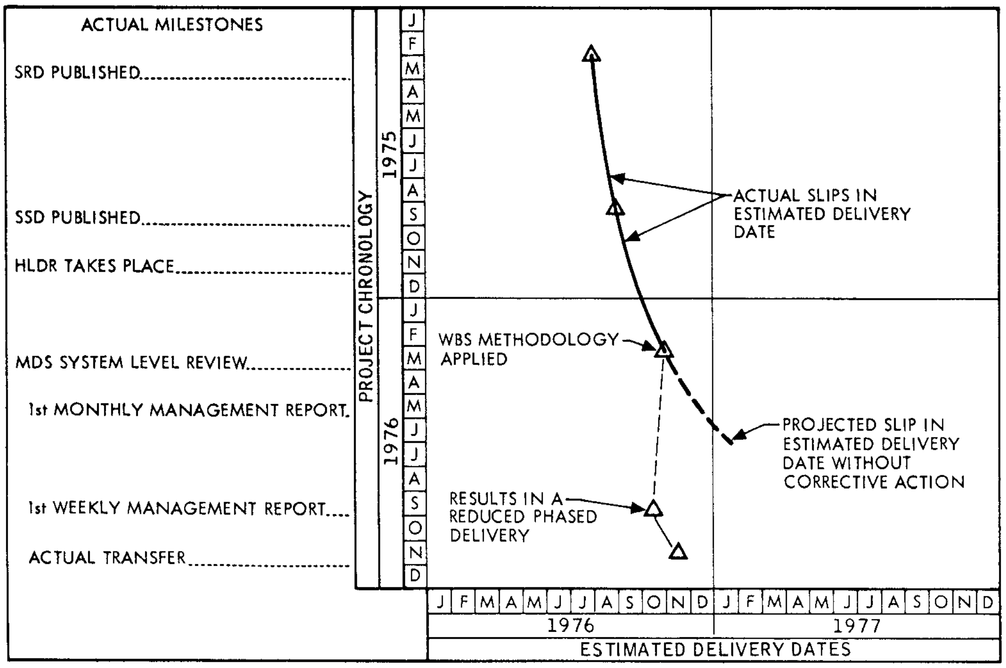

Fig. 1. Successive estimate history of DCD software delivery date

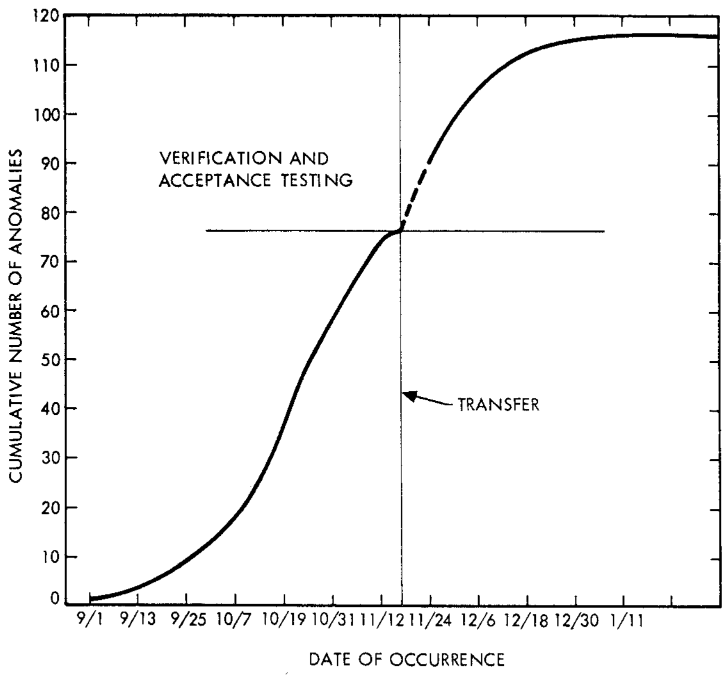

Fig. 2. DCD software anomaly history

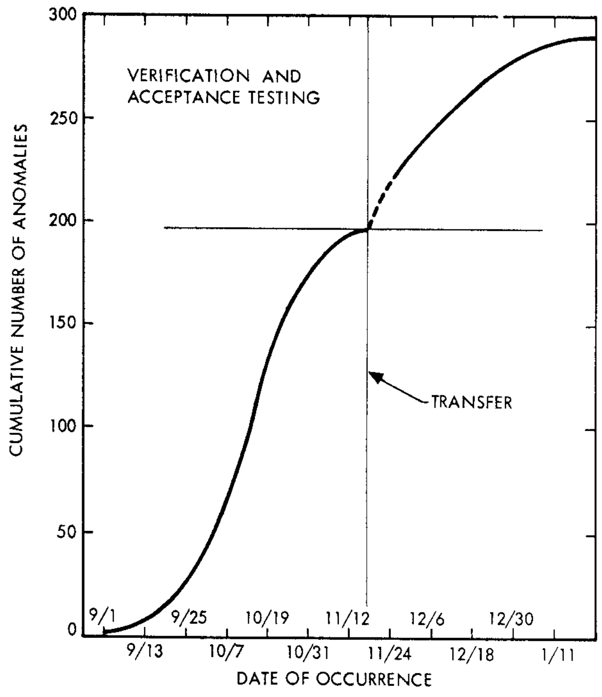

Fig. 3. MDS software anomaly history

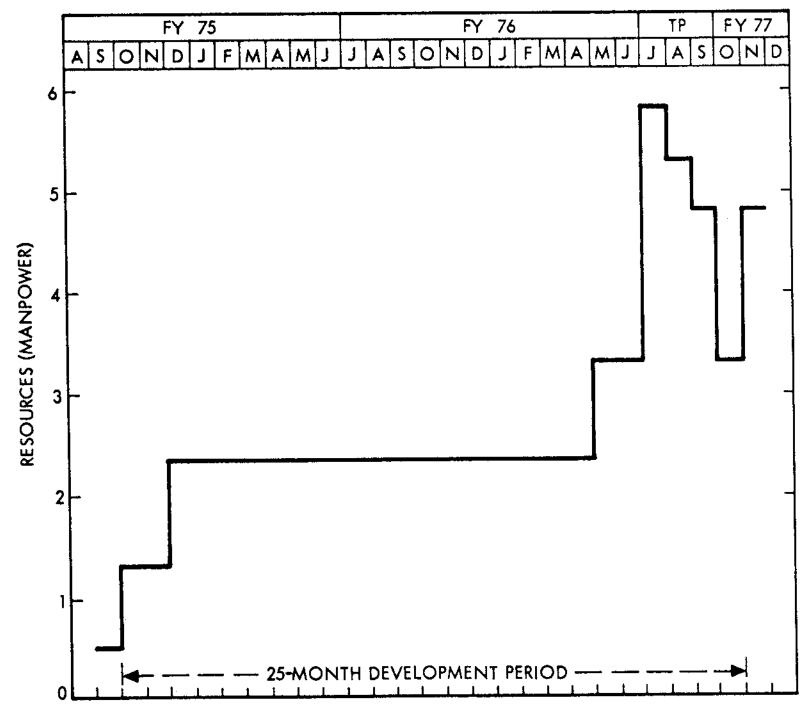

Fig. 4. Command manpower loading profile

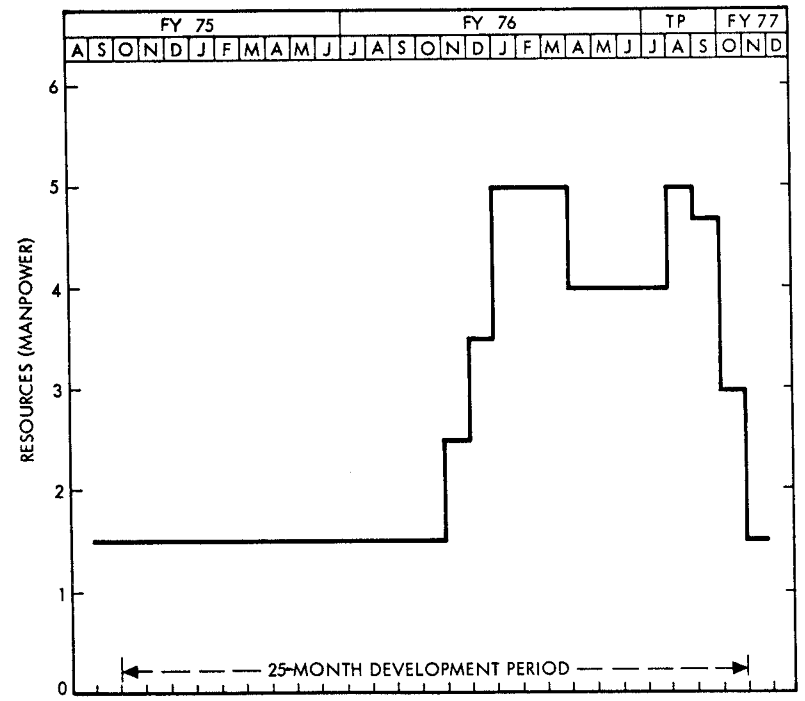

Fig. 5. CMF manpower loading profile

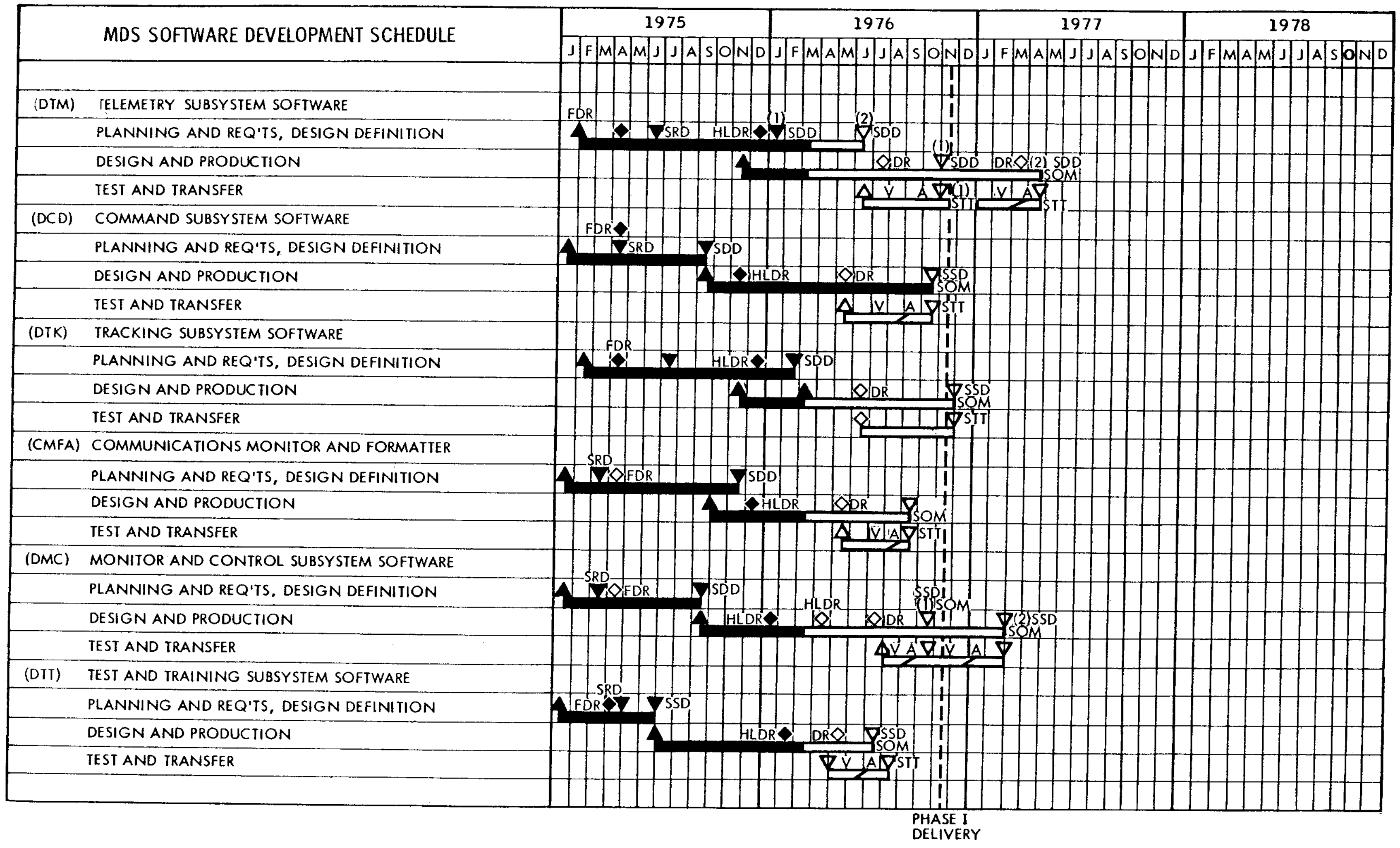

Fig. 6. MDS implementation schedule